Reasoning about Dynamic Networks of Infinite-State Processes with Global Synchronization

Ahmed Bouajjani, Yan Jurski, Mihaela Sighireanu

Reasoning about Dynamic Networks of Infinite-State Processes with Global Synchronization

Ahmed Bouajjani, Yan Jurski, and Mihaela Sighireanu

LIAFA, University of Paris 7, Case 7014, 2 place Jussieu, 75251 Paris 05, France.
{abou,jurski,sighirea}@liafa.jussieu.fr

Abstract. We propose a generic framework for reasoning about dynamic networks of infinite state processes such as counter processes, timed processes, or pushdown processes, with complex synchronization mechanisms, including global synchronization (i.e., broadcast communication). We define models for such networks, called CTN, based on Petri nets with transfer operations. Tokens (representing occurrences of processes) have attached colors over infinite domains (representing data values, clocks, stacks, etc.). We also define a (second-order) logic called CTSL allowing to express constraints on locations of tokens in the nets and on their colors. We prove that the $\exists^*\forall^*$ fragment of CTSL is decidable whenever the underlying logic for expressing constraints on colors is decidable. Moreover, we show that the same fragment is closed under *post* and *pre* image computations. These results can be used in verification such as in invariance checking. We show that our framework can be applied for reasoning about multithreaded programs with procedure calls and dynamic creation of process with global synchronization, and on dynamic programs with real-time constraints.

1 Introduction

Automated verification of modern software systems require reasoning about several complex features such as dynamic creation of concurrent threads, data manipulation, procedure calls, timing constraints, etc. For that infinite-state models must be considered allowing to capture these features, and algorithmic techniques must be designed allowing to cope with these multiple sources of infinity in the state space.

In this paper, we address the problem of defining a generic framework for automated reasoning about dynamic/parametrized networks of various classes of infinite-state processes (e.g., timed processes, processes with infinite data domains, processes with recursive procedure calls, etc). In particular, we are interested in developing a framework which can be used for the verification of multithreaded programs (written in languages such as (real-time) Java, C, etc).

In the last decade, a lot of work has been done concerning the algorithmic verification of dynamic/parametrized networks of finite-state processes (see, e.g., [20, 19, 13, 17, 3, 10]), and the verification of infinite-state models with various kinds of variables and data structures such as counters, clocks, stacks, queues, etc. (see, e.g., [1, 9, 26, 8, 6, 22, 21]). However, fewer work has been done concerning the verification of unbounded networks of infinite-state processes [4, 18, 14, 2, 7] (see related work). In a previous work [11], we have introduced a framework for reasoning about such systems

based on constrained Petri nets (CPN). Tokens in these models represent occurrences of processes (which can be dynamically created and deleted) and colors are associated with each token representing its local state (e.g., data values, age, etc). Then, to express constraints on the location of tokens and on their colors, we introduced a first-order logic over tokens and colors, called CML, which is parametrized by a logic on the color domain (e.g., Presbuger arithmetics, logic over reals, etc). We have shown in [11] that the $\exists^*\forall^*$ fragment of CML is decidable whenever the underlying color logic is, and we have also shown that this fragment is closed under computation of *post* and *pre* images. These results can be used, e.g., in Hoare-style pre-post condition reasoning, for bounded model-checking, and for checking inductive invariance of given assertions. We have shown that this framework allows indeed to handle non trivial examples of parametrized/dynamic systems (such as a parametrized Reader-Writer lock protocol).

However, CPN models do not allow to consider global synchronization (broadcast communication) between processes. For instance, synchronization mechanisms in programming languages include broadcast primitive (such as notifyAll in Java). Also, in networks of timed systems, time progress can be seen as global synchronization since all clocks must be increased with the same amount of time. Therefore, we investigate in this paper the extension of our framework of [11] in order to take into account global synchronization. This extension is not straightforward. We extend CPN models by allowing, in parallel with usual transitions, transfer operations moving simultaneously all tokens from a place to another one (may be under some constraint on their colors, and by applying a uniform transformation to the color of each of them). The new class of models is called Constrained Transfer Nets (CTN). However, the main problem for extending the previous framework is that the logic CML cannot express the effect of a transfer operation (which means that it is not closed under post and pre computations for CTN models). Actually, to express the effect of transfer operations, the (first-order) CML logic must be extended to a second-order logic where it is possible to refer to sets of tokens, and also to coloring mappings between tokens and colors. The new logic is called CTSL (for Colored Token Sets Logic). Then, we show that analogous decidability and closure properties of CML holds also for CTSL. We prove that the satisfiability problem of this new logic is decidable for the fragment $\exists^*\forall^*$ (where quantifications are this time about individuals as well as on sets) whenever the used color logic is decidable. Also, the same fragment is shown to be closed under computation of *post* and *pre* images. As previously, these results allow to automatize the reasoning about CTN models (in particular, for checking inductive invariance properties).

Then, to demonstrate the genericity and the applicability of our framework, we show that two important classes of dynamic systems can be handled as instance of this framework. First, we consider boolean Java programs with dynamic creation of concurrent threads and procedure calls, where threads are synchronized on their access to shared objects. Reasoning about such programs and their properties is quite complex [25]. We provide a modeling of this class of programs using CTN. Basically, these programs can be seen as networks of synchronizing pushdown systems. Therefore, in this case colors attached to tokens in CTN models are words representing the stack contents of the corresponding process. For that, we define an appropriate color logic, called SRC, for expressing constraints on vectors of stack contents and we prove that it is decidable (by

showing that each formula can be translated into a finite transducer). Then, we show that important properties relevant for these programs can be expressed and checked in our framework (such as deadlock freeness when threads share multiple resources, mutual exclusion, etc). Interestingly, both modeling of the considered systems and the expression of their properties involve constraints on stack contents which cannot be expressed using a finite amount of information. This is due to the fact that unbounded nesting of procedure calls makes necessary to handle the whole content of the stack. For instance, nesting of procedure calls may generate unbounded nesting of acquire-release blocks. Therefore, to release a lock needs checking the content of the stack. Notice that the ability of expressing constraints on stack contents is also useful for other applications such as in access control issues (e.g, to model stack inspection mechanisms).

Furthermore, we consider the case of real-time Java multithreaded programs which can be modeled as dynamic networks of timed automata (with control state invariants and urgent transitions). We show a CTN modeling of these systems using transfer operation to encode time elapsing transitions. Urgency can be handled in our framework due to the fact that the allowed logical language for expressing transition guards is closed under negation. As an illustrating example, we have carried out a parametrized proof of the Fisher mutual exclusion protocol where parameters are (1) the number of processes, and (2) the delays spent at each control location. We prove that the protocol is correct under a condition between these delays, for any number of processes. Previous works on the parametric verification of this protocol either consider fixed delays [4] or a fixed number of processes [6].

For lack of space, proofs and presentation of examples are given in [12].

Related work: The use of unbounded Petri nets as models for parametrized networks of processes has been proposed in many existing works such as [23, 20, 19]. However, these works consider networks of *finite-state* processes and do not address the issue of manipulating infinite data domains. The extension of this idea to networks of infinite-state processes has been addressed in few works [4, 18, 14, 2]. In [4], Abdulla and Jonsson consider the case of networks of 1-clock timed systems and show, using the theory of well-structured systems and well quasi orderings [1, 22], that the verification problem for a class of safety properties is decidable. Their approach has been extended in [18, 14] to a particular class of multiset rewrite systems with constraints (see also [2] for recent developments of this approach). Our modeling framework is inspired by these works. However, while they address the issue of deciding the verification problem of safety properties (by reduction to the coverability problem) for specific classes of systems, we consider in our work a general framework, allowing to deal in a generic way with various classes of systems, where the user can express assertions about the configurations of the system, and check automatically that they hold (using post-pre reasoning and inductive invariant checking) or that they do not hold (using bounded reachability analysis). Our framework allows to reason automatically about systems which are beyond the scope of the techniques proposed in [4, 18, 14, 2].

In a series of papers, Pnueli et al. developed an approach for the verification of parameterized systems combining abstraction and proof techniques (see, e.g., [7]). We propose here a different framework for reasoning about these systems. In [7], the authors consider a logic on (parametric-bound) *arrays* of integers, and they identify a

fragment of this logic for which the satisfiability problem is decidable. In this fragment, they restrict the shape of the formula (quantification over indices) to formulas in the fragment $\exists^*\forall^*$ similarly to what we do, and also the class of used arithmetical constraints on indices and on the associated values. In a recent work by Bradley and al. [15], the satisfiability problem of the logic of unbounded arrays with integers is investigated and the authors provide a new decidable fragment, which is incomparable to the one defined in [7], but again which imposes similar restrictions on the quantification alternation in the formulas, and on the kind of constraints that can be used. In contrast with these works, we consider a logic on *multisets* of elements with *any* kind of associated data values, provided that the used theory on the data domain is decidable. For instance, we can use in our logic general Presburger constraints whereas [7] and [15] allow limited classes of constraints. On the other hand, we cannot specify faithfully unbounded arrays in our decidable fragment because formulas of the form $\forall\exists$ are needed to express that every non extremal element has a successor/predecessor. Nevertheless, for the verification of safety properties and invariant checking, expressing this fact is not necessary, and therefore, it is possible to handle in our framework all usual examples of parametrized systems (such as mutual exclusion protocols) considered in the works mentioned above.

2 Colored Token Sets Logic

2.1 Preliminaries

Consider an enumerable set of *tokens* and let us identify this set with the set of natural numbers $\mathbb{N}$. Intuitively, tokens represent occurrences of (parallel) threads. We assume that tokens may have colors corresponding for instance to data values attached to the corresponding threads. Let $\mathbb{C}$ be a (potentially infinite) *token color domain*. Examples of color domains are the set of natural numbers $\mathbb{N}$ and the set of real numbers $\mathbb{R}$.

To express constraints on token colors, we use first-order logics over the considered color domains. In the sequel we refer to such logics as *color logics*. Presburger arithmetics $\mathsf{PA} = (\mathbb{N}, \{0, 1, +\}, \{\leq\})$ is an example of such a logic. It is well known that the satisfiability problem of Presburger arithmetics is decidable. First-order theory of reals $\mathsf{FO}_{\mathbb{R}} = (\mathbb{R}, \{0, 1, +, \times\}, \{\leq\})$ is also a decidable logic which can be used as a color logic. We introduce in section 4 another example of color logic allowing to reason about vectors of stacks (which is useful in modeling programs with procedure calls).

2.2 Syntax and semantics of CTSL

We define hereafter the syntax of the *Colored Token Sets Logic*, $\mathtt{CTSL}(L)$, which is parametrized with a color logic L. Then, let $L = (\mathbb{C}, \Omega, \Xi)$ be the first-order logic over the color domain $\mathbb{C}$, the set of functions Ω, and the set of relations Ξ. In the sequel, we omit the parameter of CTSL when its specification is not necessary.

Let T be the set of *token variables* and let C be the set of *color variables*. Colors are associated with tokens through *coloring functions*, i.e., mappings from tokens to colors. Let Γ be a set of *token coloring variables*. We assume that T, C, and Γ are disjoint.

Then, the set of CTSL terms (called *token color terms*) is given by the grammar:

$$t ::= z \mid \gamma(x) \mid \omega(t_1, \ldots, t_n)$$

where $z \in C$, $\gamma \in \Gamma$, $x \in T$, and $\omega \in \Omega$.

We consider a set of second order variables $\mathcal{X}$, called *token set variables*, representing sets of tokens. Then, the set of CTSL formulas is given by:

$$\varphi ::= x = y \mid X(x) \mid \xi(t_1, \ldots, t_m) \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists z.\ \varphi \mid \exists x.\ \varphi \mid \exists X.\ \varphi \mid \exists \gamma.\ \varphi$$

where $x, y \in T$, $z \in C$, $\xi \in \Xi$, $X \in \mathcal{X}$, $\gamma \in \Gamma$, and $t_1, \ldots, t_m$ are token color terms. Boolean connectives such as conjunction ($\wedge$), implication ($\Rightarrow$), and universal quantification ($\forall$) can be defined in terms of $\neg$, $\vee$, and $\exists$. We also use $\exists x \in X.\ \varphi$ (resp. $\forall x \in X.\ \varphi$) as an abbreviation of the formula $\exists x.\ X(x) \wedge \varphi$ (resp. $\forall x.\ X(x) \Rightarrow \varphi$). Notice that the set of terms (resp. formulas) of L is included in the set of terms (resp. formulas) of $\texttt{CTSL}(L)$.

The notions of free/bound occurrences of variables in formulas and the notions of closed/open formulas are defined as usual in second-order logics. In the sequel, we assume w.l.o.g. that in every formula, each variable is quantified at most once.

A valuation of token (resp. color , token coloring) variables is a mapping in $[T \to \mathbb{N}]$ (resp. $[C \to \mathbb{C}]$, $[\Gamma \to (\mathbb{N} \to \mathbb{C})]$). A valuation of token set variables is a mapping in $[\mathcal{X} \to 2^{\mathbb{N}}]$. Then, we define a *token sets coloring* to be a pair $\langle \nu, \mu \rangle$ where ν (resp. μ) is valuation of the token set variables (resp. token coloring variables).

We define a satisfaction relation between set colorings and CTSL formulas. For that, we need first to define the semantics of CTSL terms. Given valuations $\theta \in [T \to \mathbb{N}]$, $\delta \in [C \to \mathbb{C}]$, and $\mu \in [\Gamma \to (\mathbb{N} \to \mathbb{C})]$, we define a mapping $\langle\langle \cdot \rangle\rangle_{\theta,\delta,\mu}$ which associates with each color term a value in $\mathbb{C}$:

$$\begin{aligned}
\langle\langle z \rangle\rangle_{\theta,\delta,\mu} &= \delta(z) \\
\langle\langle \gamma(x) \rangle\rangle_{\theta,\delta,\mu} &= \mu(\gamma)(\theta(x)) \\
\langle\langle \omega(t_1, \ldots, t_n) \rangle\rangle_{\theta,\delta,\mu} &= \omega(\langle\langle t_1 \rangle\rangle_{\theta,\delta,\mu}, \ldots, \langle\langle t_n \rangle\rangle_{\theta,\delta,\mu})
\end{aligned}$$

We define inductively the satisfaction relation $\models_{\theta,\delta}$ between token sets coloring $\langle \nu, \mu \rangle$ and CTSL formulas as follows:

$$\begin{aligned}
\langle \nu, \mu \rangle \models_{\theta,\delta} \xi(t_1, \ldots, t_m) &\text{ iff } \xi(\langle\langle t_1 \rangle\rangle_{\theta,\delta,\mu}, \ldots, \langle\langle t_m \rangle\rangle_{\theta,\delta,\mu}) \\
\langle \nu, \mu \rangle \models_{\theta,\delta} X(x) &\text{ iff } \theta(x) \in \nu(X) \\
\langle \nu, \mu \rangle \models_{\theta,\delta} x = y &\text{ iff } \theta(x) = \theta(y) \\
\langle \nu, \mu \rangle \models_{\theta,\delta} \neg\varphi &\text{ iff } \langle \nu, \mu \rangle \not\models_{\theta,\delta} \varphi \\
\langle \nu, \mu \rangle \models_{\theta,\delta} \varphi_1 \vee \varphi_2 &\text{ iff } \langle \nu, \mu \rangle \models_{\theta,\delta} \varphi_1 \text{ or } \langle \nu, \mu \rangle \models_{\theta,\delta} \varphi_2 \\
\langle \nu, \mu \rangle \models_{\theta,\delta} \exists x.\ \varphi &\text{ iff } \exists t \in \mathbb{N}.\ \langle \nu, \mu \rangle \models_{\theta[x \leftarrow t],\delta} \varphi \\
\langle \nu, \mu \rangle \models_{\theta,\delta} \exists z.\ \varphi &\text{ iff } \exists c \in \mathbb{C}.\ \langle \nu, \mu \rangle \models_{\theta,\delta[z \leftarrow c]} \varphi \\
\langle \nu, \mu \rangle \models_{\theta,\delta} \exists X.\ \varphi &\text{ iff } \exists N \subset \mathbb{N}.\ \langle \nu[X \to N], \mu \rangle \models_{\theta,\delta} \varphi \\
\langle \nu, \mu \rangle \models_{\theta,\delta} \exists \gamma.\ \varphi &\text{ iff } \exists f \in [\mathbb{N} \to \mathbb{C}].\ \langle \nu, \mu[\gamma \leftarrow f] \rangle \models_{\theta,\delta} \varphi
\end{aligned}$$

For every formula φ, we define $[\![\varphi]\!]_{\theta,\delta}$ to be the set of sets coloring $\langle \nu, \mu \rangle$ such that $\langle \nu, \mu \rangle \models_{\theta,\delta} \varphi$. A formula φ is *satisfiable* iff there exist valuations θ and δ s.t. $[\![\varphi]\!]_{\theta,\delta} \neq \emptyset$. If φ is satisfiable for a subset N of $\mathbb{N}$, i.e., when the valuations ν, μ, and θ are restricted to N, we use the notation $\langle \nu, \mu \rangle \models^{N}_{\theta,\delta} \varphi$.

2.3 Syntactical forms and fragments

Prenex normal form: A formula is in *prenex normal form* (PNF) if it is of the form

$$Q_1 y_1 Q_2 y_2 \dots Q_m y_m.\ \varphi$$

where (1) $Q_1, \dots, Q_m$ are (existential or universal) quantifiers, (2) $y_1, \dots, y_m$ are variables in $T \cup C \cup \mathcal{X} \cup \Gamma$, and φ is a quantifier-free formula. For every formula φ in CTSL, there exists an equivalent formula φ' in prenex normal form.

Quantifier alternation hierarchy: We consider two families $\{\Sigma_n\}_{n\geq 0}$ and $\{\Pi_n\}_{n\geq 0}$ of fragments of CTSL defined according to the alternation depth of existential and universal quantifiers in their PNF:

- $\Sigma_0 = \Pi_0$ be the set of formulas in PNF where all quantified variables are in C,
- For $n \geq 0$, let Σ_{n+1} (resp. Π_{n+1}) be the set of formulas $Qy_1 \dots y_m.\ \varphi$ in PNF where $y_1, \dots, y_m \in T \cup C \cup \mathcal{X} \cup \Gamma$, Q is the existential (resp. universal) quantifier $\exists$ (resp. $\forall$), and φ is a formula in Π_n (resp. Σ_n).

It is easy to see that, for every $n \geq 0$, Σ_n and Π_n are closed under conjunction and disjunction, and that the negation of a Σ_n formula is a Π_n formula and vice versa. For every $n \geq 0$, let $B(\Sigma_n)$ denote the set of all boolean combinations of Σ_n formulas. Clearly, $B(\Sigma_n)$ subsumes both Σ_n and Π_n, and is included in both Σ_{n+1} and Π_{n+1}.

2.4 Satisfiability Problem

We investigate the decidability of the satisfiability problem of the logic CTSL(L), assuming that the underlying color logic L has a decidable satisfiability problem.

Theorem 1. *If the satisfiability problem of L is decidable, then the fragment Σ_2 of* CTSL(L) *is decidable.*

Theorem 1 cannot be extended beyond the Σ_2 fragment, even for simple color logics.

Theorem 2. *([11]) The satisfiability problem is undecidable for the first-order Π_2 fragment of* CTSL(OL) *where* $OL = (\mathbb{N}, 0, \leq)$.

3 Colored Transfer Nets

We present hereafter our program models which are based on Petri nets with transfer arcs. We need first to introduce some definitions and notations. Let $\mathcal{P}$ be a finite subset of $\mathcal{X}$. Elements of $\mathcal{P}$ are called *places* and denoted by $p, q, r, s, \dots$. $\perp$ is a special elements of $\mathcal{P}$ called *nowhere place*. Let $\mathcal{P}' \subset \mathcal{X}$ be a set disjoint from $\mathcal{P}$ such that $|\mathcal{P}| = |\mathcal{P}'|$; its elements are denoted by $p', q', r', s', \dots$. Let $\mathcal{G}$ be a finite subset of C, which elements are called *global variables* and are denoted by $g, l, \dots$. Let $\mathcal{G}' \subset C$ be a set of color variables disjoint from $\mathcal{G}$ such that $|\mathcal{G}| = |\mathcal{G}'|$; its elements are denoted by $g', l', \dots$. Let $\mathcal{L}$ be a finite subset of Γ, which elements are called *local variables*. Let $\mathcal{L}' \subset \Gamma$ a set of coloring symbols disjoint from $\mathcal{L}$ and with the same size. Primed (resp. unprimed) variables are used to refer to values after (resp. before) the execution of transitions.

A *Colored Transfer Net* (CTN) is a tuple $S = (\mathcal{P}, \mathcal{G}, \mathcal{L}, L, \Delta)$ where $L = (\mathbb{C}, \Omega, \Xi)$ is a color logic and Δ is a finite set of *constrained transitions* of the form:

$$\overrightarrow{x}.\,(\overrightarrow{p} \hookrightarrow \overrightarrow{q} : \varphi) \mid \overrightarrow{y}.\,(\overrightarrow{r} \mapsto \overrightarrow{s} : \psi) \tag{1}$$

where $\overrightarrow{x}, \overrightarrow{y} \in T$, $\overrightarrow{p}, \overrightarrow{q}, \overrightarrow{s}, \overrightarrow{r} \in \mathcal{P}$, $\bot \notin \overrightarrow{r}$, $|\overrightarrow{x}| = |\overrightarrow{p}| = |\overrightarrow{q}| = n$, $|\overrightarrow{y}| = |\overrightarrow{r}| = |\overrightarrow{s}| = m$, $\overrightarrow{p} \cap \overrightarrow{r} = \emptyset$, $\overrightarrow{p} \cap \overrightarrow{s} = \emptyset$, $\overrightarrow{q} \cap \overrightarrow{r} = \emptyset$, and all places in $\overrightarrow{r}$ are disjoint. φ is a formula in $\mathtt{CTSL}(L)$ whose free variables are $\overrightarrow{x}$, global variables in $\mathcal{G} \cup \mathcal{G}'$, local variables in $\mathcal{L}$, and symbol colors of $\mathcal{L}'$ if they are applied to variables in $\overrightarrow{x}$. ψ is a formula in $\mathtt{CTSL}(L)$ whose free variables are $\overrightarrow{y}$, global variables in $\mathcal{G} \cup \mathcal{G}'$, local variables in $\mathcal{L}$, and symbol colors of $\mathcal{L}'$ if they are applied to variables in $\overrightarrow{y}$.

A CTN transition has two parts separated by $|$. The left-hand-side, called the *existential part*, corresponds to Petri nets arcs with bounded cardinality: a variable x_i of $\overrightarrow{x}$ is bound to a selected token in the corresponding place p_i of $\overrightarrow{p}$, the token selected is deleted from p_i and put in the corresponding place q_i of $\overrightarrow{q}$. Formula φ gives the selection criterion for tokens in $\overrightarrow{p}$ and how the colors of these tokens are changed (using variables in $\mathcal{L}'$ applied to $\overrightarrow{x}$). Also, φ describe changes on global variables by defining the value of $\mathcal{G}'$. The right-hand-side of a rule, called the *universal part*, corresponds to Petri nets arcs with unbounded cardinality (*transfer* arcs): *all* tokens of each place in $\overrightarrow{r}$ satisfying the selection criterion given by ψ are transfered to the corresponding place in $\overrightarrow{s}$ and theirs colors are changed according to ψ. (Formula ψ uses variables in $\overrightarrow{y}$ to refer to each transfered token.)

3.1 Transitions as formulas

We associate with each transition τ of the form (1) a $\mathtt{CTSL}(L)$ formula:

$$\begin{aligned}
&\mathit{reach}_\tau(\mathcal{P}, \mathcal{P}', \mathcal{G}, \mathcal{G}', \mathcal{L}, \mathcal{L}') = \exists x_1, ..., x_n.\ \exists Y_1, ..., Y_m.\\
&\quad (\wedge_{i=1}^{n} p_i(x_i) \wedge (\forall t.\ (\wedge_{j=1}^{m} \neg Y_j(t)) \Rightarrow \wedge_{i=1}^{n} (q_i'(t) \Leftrightarrow (q_i(t) \vee t = x_i)))) \ \wedge\ \varphi\\
&\quad \wedge\ (\forall t.\ ((\wedge_{j=1}^{m} \neg Y_j(t)) \wedge (\wedge_{i=1}^{n} t \neq x_i)) \Rightarrow (\wedge_{a \in \mathcal{L}} a'(t) = a(t) \wedge \wedge_{p \in \mathcal{P}} p(t) \Leftrightarrow p'(t)))\\
&\quad \wedge\ \wedge_{j=1}^{m} (\forall y.\ (r_j'(y) \Rightarrow r_j(y)) \wedge (Y_j(y) \Rightarrow r_j(y)) \wedge\\
&\qquad\qquad (r_j(y) \Rightarrow ((r_j'(y) \wedge \neg Y_j(y)) \vee (\neg r_j'(y) \wedge Y_j(y)))))\\
&\quad \wedge\ (\forall y.\ \wedge_{j=1}^{m} (s_j'(y) \Leftrightarrow (s(y) \vee Y_j(y))))\\
&\quad \wedge\ (\forall y_1, ..., y_m.\ (\wedge_{j=1}^{m} Y_j(y_j)) \Leftrightarrow \psi)
\end{aligned}$$

Token variables $x_1, ..., x_n$ represent moved token by the existential part of the rule, and the token sets variables $Y_1, ..., Y_m$ are used to describe the transfered tokens by the universal part. The first line of the formula models the moving of tokens by the existential part of the transition, and imposes the constraints φ on the moved tokens. The second line says that the tokens which are not involved in the transition keep the same places and local colors. The third and fourth line says that r_j is partitioned between sets Y_j and r_j'. The fifth line says that the set of tokens Y_j is added to the corresponding target place s_j' after the transfer. The last line constrains the sets Y_j and their color change by ψ.

Given a fragment Θ of CTSL, we denote by $\mathtt{CTN}[\Theta]$ the class of CTN where each transition corresponds to a formula in the fragment Θ. Due to the (un)decidability results of section 2.4, we focus in the sequel on the classes $\mathtt{CTN}[\Sigma_2]$ and $\mathtt{CTN}[\Sigma_1]$.

3.2 Semantics

Configurations of CTNs, called *colored markings*, are triples $\langle M, \mu, \delta \rangle$ where (1) $M \in [\mathbb{N} \to \mathcal{P}]$ is an application, called *marking*, which associates to each token a place, (2) $\mu \in [\mathcal{L} \to (\mathbb{N} \to \mathbb{C})]$ is a valuation of local variabes in $\mathcal{L}$, and (3) $\delta \in [\mathcal{G} \to \mathbb{C}]$ is a valuation of global variables in $\mathcal{G}$.

For a marking M, let ν_M be a mapping in $[\mathcal{P} \to 2^{\mathbb{N}}]$ such that for all $p \in \mathcal{P}$ $\nu_M(p) = \{t \in \mathbb{N} \mid M(t) = p\}$. For a mapping ν_M (resp. μ, δ), we denote by $\widehat{\nu_M}$ (resp. $\widehat{\mu}$, $\widehat{\delta}$) the mapping obtained by replacing each variable in $\mathcal{P}$ (resp. $\mathcal{L}$, $\mathcal{G}$) by its corresponding element in $\mathcal{P}'$ (resp. $\mathcal{L}'$, $\mathcal{G}'$). The union of mappings defined on disjoint domains is denoted by the $\oplus$ operator.

Constrained transitions of CTN define a transition relation $\to_S$ between CTN configurations as follows: For every two configurations $\langle M_1, \mu_1, \delta_1 \rangle$ and $\langle M_2, \mu_2, \delta_2 \rangle$, we have $\langle M_1, \mu_1, \delta_1 \rangle \to_S \langle M_2, \mu_2, \delta_2 \rangle$ iff there exists a constrained transition $\tau \in \Delta$ such that:

$$\langle \nu_{M_1} \oplus \widehat{\nu_{M_2}}, \mu_1 \oplus \widehat{\mu_2} \rangle \models_{\emptyset, \delta_1 \oplus \widehat{\delta_2}} \mathit{reach}_\tau(\mathcal{P}, \mathcal{P}', \mathcal{G}, \mathcal{G}', \mathcal{L}, \mathcal{L}')$$

Given $\mathcal{M} = \langle M, \mu, \delta \rangle$, let $\mathsf{post}_S(\mathcal{M}) = \{\mathcal{M}' \,:\, \mathcal{M} \to_S \mathcal{M}'\}$ be the set of its immediate successors, and let $\mathsf{pre}_S(\mathcal{M}) = \{\mathcal{M}' \,:\, \mathcal{M}' \to_S \mathcal{M}\}$ be the set of its immediate predecessors. These definitions can be generalized to sets of configurations.

3.3 Computing post and pre images

We show hereafter the following closure property of CTSL fragments under the computation of immediate successors and predecessors for CTNs.

Theorem 3. *Let $S = (\mathcal{P}, \mathcal{G}, \mathcal{L}, L, \Delta)$ be a $\mathtt{CTN}[\Sigma_n]$, for $n \in \{1, 2\}$. Then, for every formula ψ in the fragment Σ_n of the logic $\mathtt{CTSL}(L)$ with free variables in $\mathcal{P} \cup \mathcal{G} \cup \mathcal{L}$, it is possible to construct two formulas ψ_{post} and ψ_{pre} in the same fragment Σ_n such that $[\![\psi_{\mathsf{post}}]\!] = \mathsf{post}_S([\![\psi]\!])$ and $[\![\psi_{\mathsf{pre}}]\!] = \mathsf{pre}_S([\![\psi]\!])$.*

Proof. Let $\psi(\mathcal{P}, \mathcal{G}, \mathcal{L})$ be a formula of $\mathtt{CTSL}(L)$, and let τ be a transition in Δ. We define hereafter the formulas ψ_{post} and ψ_{pre} for this single transition. The generalization to the set of all transitions is straightforward. The construction of the formulas ψ_{post} and ψ_{pre} is trivial due to the fact that our logic allows to use quantification over places and coloring symbols (representing with local variables). In the following, we use the notation $\exists \mathcal{P}$ for $\exists p_1, ..., p_n$ with $\mathcal{P} = \{p_1, ..., p_n\}$. Also, $\phi[\mathcal{P}' \leftarrow \mathcal{P}]$ means $\phi[p'_1 \leftarrow p_1, ..., p'_n \leftarrow p_m]$. Similar notations are used for sets $\mathcal{G}$ and $\mathcal{L}$. Then ψ_{post} and ψ_{pre} are defined by:

$$\psi_{\mathsf{post}} = (\exists \mathcal{P}\ \exists \mathcal{G}\ \exists \mathcal{L}.\ \psi \wedge \mathit{reach}_\tau)[\mathcal{P}' \leftarrow \mathcal{P}][\mathcal{G}' \leftarrow \mathcal{G}][\mathcal{L}' \leftarrow \mathcal{L}]$$
$$\psi_{\mathsf{pre}} = (\exists \mathcal{P}'\ \exists \mathcal{G}'\ \exists \mathcal{L}'.\ \psi[\mathcal{P} \leftarrow \mathcal{P}'][\mathcal{G} \leftarrow \mathcal{G}'][\mathcal{L} \leftarrow \mathcal{L}'] \wedge \mathit{reach}_\tau)$$

It is easy to see that if ψ and reach_τ are in a fragment Σ_n, for any $n \geq 1$, then both of the formulas ψ_{post} and ψ_{pre} are also in the same fragment Σ_n.

See in [12] an example of post-image computation.

3.4 Invariance checking

The results above allow to perform various kinds of analysis for CTN, e.g., Hoare-style post-pre condition reasoning, bounded reachability analysis, or invariance checking problem. We detail here the case of *inductive invariance checking problem*. An instance of such a problem is given by a triple $(Init, Inv, Aux)$ of sets of configurations, where *Init* is a set of initial configurations, *Inv* is the invariant to be proved, and *Aux* is an auxilliary invariant. It consists in deciding whether (1) $Init \subseteq Aux$, (2) $Aux \subseteq Inv$, and (3) *Aux* is an inductive invariant, i.e., that $\mathsf{post}(Aux) \subseteq Aux$. The following result follows from Theorem 3, Theorem 1, and the previous theorem.

Theorem 4. *Let S be a* $\mathtt{CTN}[\Sigma_2]$. *The inductive invariance checking problem is decidable for every instance* $([\![\psi_{Init}]\!], [\![\psi]\!], [\![\psi']\!])$ *where* $\psi_{Init} \in \Sigma_2$, *and* $\psi, \psi' \in B(\Sigma_1)$.

In the full version [12], we give an example illustrating the use of this theorem.

4 Boolean Multithread Programs with Infinite Control

We consider here boolean Java programs with dynamic creation of threads, recursive methods, and threads synchronized on a set of shared objects. We show how to verify the absence/presence of deadlock in such programs by modeling and reasoning about the stack of each thread.

Let us first recall briefly the semantics of synchronization between threads in Java. Threads synchronize their access to global shared objects using a *lock* mechanism: each object has a lock that may be owned by at most one thread. A thread owns the lock of an object *o* while executing blocks of code labeled by `synchronized(`*o*`)` or methods of the object *o* declared as `synchronized`. A synchronized method or block of code can contain calls to other synchronized methods of the same or of a different object. To avoid starvation or deadlocks, threads can suspend their execution and free the lock of the object it owns by calling the `wait` method. While owning a lock on object *o*, a thread can awake one or all threads suspended on *o* using methods `notify` resp. `notifyAll`. The latter method is a mean of global synchronization of threads in Java. The call of these methods is not blocking and the caller continues to own the lock of *o*. The awakened threads are now competing for the owning of the lock on *o* with other threads wanting to lock *o*. An awakened thread continues its executon with the statement after the `wait` call.

To obtain the model of a Java program of this class, we start with the *Interprocedural Control Flow Graph* (ICFG) of the program. The ICFG arcs are labeled by actions like: call of an user-defined method on an object, call of a pre-defined method (`wait`, `notify`, `notifyAll`) on a shared object, and return from a method. Wlog, we consider that `synchronized` code is present only in user-defined methods, and this information is available in the ICFG. We show in this section how to translate the ICFG into a CTN. The color logic used is the SRC logic presented below, which allows to model and reason about the stacks of threads. (See example in [12].)

Synchronous Rational Constraints Logic: Let Σ be a finite alphabet, and $\mathsf{Reg}(\Sigma)$ be the class of regular languages over Σ. Then, $\mathsf{SRC} = (\Sigma^*, \{\lambda u.\, u \cdot a \,:\, a \in \Sigma\}, \{\leq_{\mathsf{pref}}, =_\ell\} \cup \{R(\cdot) \,:\, R \in \mathsf{Reg}(\Sigma)\})$ where:

- $\leq_{\mathsf{pref}}$ is the prefix ordering, i.e., for every $u,v \in \Sigma^*$. $u \leq_{\mathsf{pref}} v$ iff $\exists w.\ v = uw$,
- $=_\ell$ is the length equality predicate i.e., for every $u,v \in \Sigma^*$, $u =_\ell v$ iff $|u| = |v|$, and
- for every $R \in \mathsf{Reg}(\Sigma)$, $R(\cdot)$ is the unary predicate s.t. $\forall u \in \Sigma^*$, $R(u)$ iff $u \in R$.

Notice that the equality predicate can be defined using the relations $\leq_{\mathsf{pref}}$ and $=_\ell$: $u = v$ is equivalent to $u \leq_{\mathsf{pref}} v \wedge u =_\ell v$.

Theorem 5. *The satisfiability problem of* SRC *is decidable.*

Modeling method calls: Places are associated with nodes (control points) of the ICFG; a token in a place represents a thread ready to execute the statements on the outgoing edges. The coloring symbol α attached to tokens represents the stack of the thread. Stack values are works ont the alphabet Σ which contains the set of all control points in the ICFG. Constrained transitions of CTN are obtained from edges of the ICFG as follows. An edge $p \xrightarrow{\mathsf{call}P()} q$ where P is a user-defined method with starting node $start_P$ is modeled by the transition $x.\ p \hookrightarrow start_P\ :\ \alpha'(x) = \alpha(x).q$. An edge $exit_P \xrightarrow{\mathsf{return}} q$ representing the return from method P is modeled by the transition $x.\ exit_P \hookrightarrow p\ :\ \alpha'(x).p = \alpha(x)$. Finally, an edge $p \rightarrow q$ representing change of control point p to q inside a method is represented by the transition $x.\ p \hookrightarrow q\ :\ \alpha'(x) = \alpha(x)$. The basic model above may be adapted to store in the stack additional information about the methods called and not exited by the thread, as we do in the following.

Modeling synchronization: To model ICFG edges including call to synchronization methods, we first create for each object o used as a lock (i.e., for which there exists a statement "`o.m()`" where `m` is a synchronized method) three places: (1) $lock_o$ models the lock of the object: at most one token is present; (2) $wait_o$ stores tokens representing threads competing to obtain the lock of o; (3) $susp_o$ stores tokens representing threads which are self-suspended by the execution of an "`wait()`" statement while owning the lock of o. To the stack alphabet Σ is added the set of objects used as locks.

An ICFG edge $p \xrightarrow{\mathsf{call}\ o.m()} q$ with m being a synchronized method is modeled by the following four transitions:

$$
\begin{array}{lll}
m_o^1: & (x,y).\ (p, lock_o) \hookrightarrow (start_m, \bot) & :\ \alpha'(x) = \alpha(x).q.o \\
m_o^2: & x.\ p \hookrightarrow start_m & :\ \alpha(x) \in (\Sigma^*.o.\Sigma^*) \wedge \alpha'(x) = \alpha(x).q.o \\
m_o^3: & x.\ p \hookrightarrow wait_o & \\
 & & :\ \neg(\exists y.\ lock_o(y)) \wedge \neg(\alpha(x) \in (\Sigma^*.o.\Sigma^*)) \wedge \alpha'(x) = \alpha(x).q.start_m \\
m_o^4: & (x,y).\ (wait_o, lock_o) \hookrightarrow (q, \bot) & :\ \exists w.\ \alpha(x) = w.q \wedge \alpha'(x) = w.o
\end{array}
$$

The first transition is the case of an available lock. Then, the lock is taken and the thread starts procedure m and puts in its stack the information about having the lock of object o. The second transition is the case where the thread is already owning the lock of o, so it starts immediately the execution of method m (object o is added on stack for uniformity). The third transition shows the case where the lock is not available and it is not owned by the thread. The thread is sent into the $wait_o$ place and the stack stores the return address and the start address of the called method. The fourth transition shows that a lock may be taken by a thread in the waiting state when it is available. Like for the first transition, the information about owning lock of o is put on the stack.

The transitions below show respectively the modeling of the call of methods `wait`, `notify`, and `notifyAll` in a synchronized method of object o:

$$\begin{array}{lll} wait_o: & (x,y).\ (p,\bot) \hookrightarrow (susp_o, lock_o) & : \exists w.\ \alpha(x) = w.o \wedge \alpha'(x) = w.q \\ notify_o: & (x,y).\ (p, susp_o) \hookrightarrow (q, wait_o) & : \alpha(x) = \alpha'(x) \wedge \alpha(y) = \alpha'(y) \\ notifyAll_o: & x.\ p \hookrightarrow q & : \alpha(x) = \alpha'(x) \\ & \mid\ y.\ susp_o \mapsto wait_o & : \alpha(y) = \alpha'(y) \end{array}$$

Finally, the return from a synchronized method is modeled by the following transition:

$$\begin{array}{lll} sret_1: & (x,y).\ (exit_m, \bot) \hookrightarrow (q, lock_o) & : \exists w.\ \alpha(x) = w.o \wedge w = \alpha'(x).q \wedge \neg(w \in (\Sigma^*.o.\Sigma^*)) \\ sret_2: & x.\ exit_m \hookrightarrow q & : \exists w.\ \alpha(x) = w.o \wedge w = \alpha'(x).q \wedge (w \in (\Sigma^*.o.\Sigma^*)) \end{array}$$

where the lock of the object is freed only if the thread has not called the method from another synchronized method of the object.

The property of deadlock freedom can be expressed as a property of the stack associated with each thread. For example, the deadlock is avoided if a thread trying to acquire the lock of an object o by calling a synchronized method of this object at control point l cannot be delayed by another thread owning the lock of this object:

$$\forall t, u.\ l(t) \implies \neg(\alpha(u) \in \Sigma^*.o.\Sigma^*)$$

`CTSL`(SRC) allows also to express properties about the order in which the objects are locked (or methods are called) by the thread. For example, the property saying that all threads at location ℓ have first acquired o_1 and then o_2 can be express as follows:

$$\forall t.\ l(t) \implies \alpha(u) \in (\Sigma - \{o_1, o_2\})^*.o_1.(\Sigma - \{o_2\})^*.o_2.(\Sigma - \{o_1\})^*$$

5 Multithreaded Programs with Timed Constraints

We consider here real-time Java programs with dynamic creation of threads (without procedure calls). Such programs can be modeled as dynamic networks of timed automata [5]. We show how to define such networks in our framework. We consider that clocks variables range over the domain of reals. Then, the color logic L can be the first-order theory of reals $\mathsf{FO}_{\mathbb{R}} = (\mathbb{R}, \{0,1,+,\times\}, \{\leq\})$ or its subfragments (e.g., linear first-order theory).

The *global and local clocks* used in the network aretranslated into variables in $\mathcal{G}$ resp. $\mathcal{L}$. We suppose that $\mathcal{G}$ contains a variable t measuring the global time. The translation of automata locations is immediate into places.

Then, we model a *discrete transition* from location ℓ_1 to location ℓ_2 whose guard is ϕ and which *resets* local clock c by the transition:

$$x.\ \ell_1 \hookrightarrow \ell_2 : \phi \wedge c'(x) = 0$$

In timed automata, *time elapsing* increases all clocks by the same amount of time. We model such it by a unique constrained transition changing uniformly the global and local colors of threads in places $\overrightarrow{p}$ where the time can elapse:

$$\hookrightarrow : \exists z.\ 0 < z \wedge t' = t + z \mid \overrightarrow{y}.\ \overrightarrow{p} \mapsto \overrightarrow{p} : \bigwedge_{\gamma \in \mathcal{L}} \gamma'(\overrightarrow{y}) = \gamma(\overrightarrow{y}) + t' - t \wedge \bigwedge_{c \in \mathcal{G}} c' = c + t' - t$$

Time invariants are associated with locations and represent constraints which must be satisfied while time elapses at these locations. Invariants used in timed automata can be expressed are formulas in the Σ_0 fragment of CTSL. Let denote by $Inv(p)$ the invariant formula obtained for the location represented by p. Then, we modify the time elapsing transition to take into account invariants:

$$\begin{array}{l} \hookrightarrow : \exists z.\ 0 < z \wedge t' = t + z \\ \quad | \quad \overrightarrow{y}.\ \overrightarrow{p} \mapsto \overrightarrow{p} : \bigwedge_{\gamma \in \mathcal{L}} \gamma'(\overrightarrow{y}) = \gamma(\overrightarrow{y}) + t' - t \wedge \bigwedge_{c \in \mathcal{G}} c' = c + t' - t \\ \qquad\qquad\qquad\qquad (\forall z'.\ 0 \leq z' \leq t' - t \implies (\bigwedge_{p_i \in \overrightarrow{p}} Inv(p_i)[\mathcal{L}(y_i) \leftarrow \mathcal{L}(y_i) + z'])) \end{array}$$

From the decidability results given by Theorem 3, it results that tractable models CTN are obtained if time invariants are at most formulas in Π_1 fragment.

Urgent transitions are discrete transitions which have more priority that time passing, i.e., they must be executed immediatly. Let $\overrightarrow{x}.\ (\overrightarrow{p} \hookrightarrow \overrightarrow{q} : \phi)$ be a discrete urgent transition. Then, to express its urgency, we must modify time elapsing transition by imposing $\neg\phi$ in the guard, i.e., time can progress only if the urgent transition is not enabled:

$$\begin{array}{l} \hookrightarrow : \exists z.\ 0 < z \wedge t' = t + z \\ \quad | \quad \overrightarrow{y}.\ \overrightarrow{p} \mapsto \overrightarrow{p} : \bigwedge_{\gamma \in \mathcal{L}_0} \gamma'(\overrightarrow{y}) = \gamma(\overrightarrow{y}) + t' - t \wedge \bigwedge_{c \in \mathcal{G}} c' = c + t' - t \\ \qquad\qquad\qquad\qquad \wedge\ (\forall z'.\ 0 \leq z' \leq t' - t \implies (\bigwedge_{p_i \in \overrightarrow{p}} Inv(p_i)[\mathcal{L}(y_i) \leftarrow \mathcal{L}(y_i) + z']) \\ \qquad\qquad\qquad\qquad\qquad\qquad \wedge \neg(\exists \overrightarrow{x}.\ \overrightarrow{p}(\overrightarrow{x}) \wedge \phi[\mathcal{L}(\overrightarrow{x}) \leftarrow \mathcal{L}(\overrightarrow{x}) + z'])) \end{array}$$

From Theorem 3, it follows that tractable CTN models are obtained if guards of urgent transitions are in Σ_1 fragment of CTSL.

6 Conclusion

We have defined a generic framework for reasoning about unbounded networks of infinite-state processes. Various instances of this framework allow to deal in a uniform way with important classes of system models such as dynamic networks of counter processes, timed processes, or pushdown processes, with complex synchronization mechanisms. This is based on generic decidability and closure results for a (useful fragment of a) logic for specifying configurations of such networks which allow to reason both on the control locations of the processes and on their data. The expressiveness of our framework has been demonstrated by showing its application to reasoning about multithreaded Java programs with global synchronization, and about multithreaded programs with timing constraints.

The complexity of the decision procedures is doubly exponential in the size of formulas (more precisely, in the number of quantified variables). However, formulas we need to consider when reasoning about CTN models (assertions expressing invariants and transition guards) are of a special form. Indeed, they are usually first-order formulas which implies that post/pre computations do not generate formulas with universal second order quantification. Moreover, token variables in transition formulas are deterministically associated with places. These facts allow to reduce the complexity by an exponential factor. Furthermore, transition formulas usually introduce a small number (2 or 3) of new token variables (the number of synchronized processes). This fact reduces significantly the complexity in practice.

References

1. P. A. Abdulla, K. Cerans, B. Jonsson, and Y.-K. Tsay. General decidability theorems for infinite-state systems. In *Proc. of LICS'96*, pages 313–321, 1996.
2. P. A. Abdulla and G. Delzanno. On the Coverability Problem for Constrained Multiset Rewriting. In *Proc. of AVIS'06, Satellite workshop of ETAPS'06*, Vienna, Austria, 2006.
3. P. A. Abdulla, B. Jonsson, M. Nilsson, and M. Saksena. A Survey of Regular Model Checking. In *Proc. of CONCUR'04*, volume 3170 of *LNCS*. Springer, 2004.
4. P.A. Abdulla and B. Jonsson. Verifying networks of timed processes (extended abstract). In Bernhard Steffen, editor, *Proc. of TACAS'98*, volume 1384 of *LNCS*, pages 298–312. LNCS 1384, 1998.
5. R. Alur and D.L. Dill. A theory of timed automata. *TCS*, 126:183–235, 1994.
6. A. Annichini, E. Asarin, and A. Bouajjani. Symbolic techniques for parametric reasoning about counter and clock systems. In *Proc. of CAV'00*. LNCS 1855, 2000.
7. T. Arons, A. Pnueli, S. Ruah, J. Xu, and L.D. Zuck. Parameterized Verification with Automatically Computed Inductive Assertions. In *Proc. of CAV'01*. LNCS 2102, 2001.
8. B. Boigelot. *Symbolic Methods for Exploring Infinite State Space*. PhD thesis, Faculté des Sciences, Université de Liège, volume 189, 1999.
9. A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *Proc. of CONCUR'97*, volume 1243 of *LNCS*, pages 135–150. LNCS 1243, 1997.
10. A. Bouajjani, P. Habermehl, and T. Vojnar. Abstract Regular Model Checking. In *Proc. of CAV'04*, volume 3114 of *LNCS*. Springer, 2004.
11. A. Bouajjani, Y. Jurski, and M. Sighireanu. A generic framework for reasoning about dynamic networks of infinite-state processes. In *TACAS'07*. LNCS, 2007.
12. A. Bouajjani, Y. Jurski, and M. Sighireanu. Reasoning about dynamic networks of infinite-state processes with global synchronization. Technical Report 2007-03, LIAFA lab, January 2007. Available at `http://www.liafa.jussieu.fr/~abou/publis.html`.
13. A. Bouajjani, M. Müller-Olm, and T. Touili. Regular symbolic analysis of dynamic networks of pushdown systems. In *Proc. of CONCUR'05*, volume 3653 of *LNCS*. Springer, 2005.
14. M. Bozzano and G. Delzanno. Beyond Parameterized Verification. In *Proc. of TACAS'02*, volume 2280 of *LNCS*, Grenoble, France, 2002. Springer Pub.
15. A. R. Bradley, Z. Manna, and H. B. Sipma. What's decidable about arrays? In *Proc. of VMCAI'06*, volume 3855 of *LNCS*. Springer, 2006.
16. Christian Choffrut. The Reachability Problem Requires Exponential Space. Technical Report 62, Yale University, 1976.
17. E. M. Clarke, O. Grumberg, and S. Jha. Verifying parameterized networks. *TOPLAS*, 19(5), 1997.
18. G. Delzanno. An assertional language for the verification of systems parametric in several dimensions. *Electr. Notes Theor. Comput. Sci.*, 50(4), 2001.
19. G. Delzanno, J.-F. Raskin, and L. Van Begin. Towards the automated verification of multithreaded java programs. In *TACAS*, volume 2280 of *LNCS*, pages 173–187. Springer, 2002.
20. E. A. Emerson and K. S. Namjoshi. On model checking for non-deterministic infinite-state systems. In *LICS'98*. IEEE, 1998.
21. A. Finkel and J. Leroux. How to compose presburger-accelerations: Applications to broadcast protocols. In *Proc. of FST&TCS'02*, volume 2556 of *LNCS*. Springer, 2002.
22. A. Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256(1-2):63–92, 2001.
23. S. M. German and P. A. Sistla. Reasoning about systems with many processes. *JACM*, 39(3), 1992.

24. Leslie Lamport. A fast mutual exclusion algorithm. *ACM Transactions on Computer Systems*, 5(1):1–11, 1987.
25. Edward A. Lee. The problem with threads. Technical Report UCB/EECS-2006-1, Electrical Engineering and Computer Science University of California at Berkeley, January 2006.
26. P. Wolper and B. Boigelot. Verifying systems with infinite but regular state spaces. In *Proc. of CAV'98*, volume 1427 of *LNCS*. Springer, 1998.

A Proof of Theorem 1

We reduce the satisfiability problem of Σ_2 formulas in $\mathtt{CTSL}(L)$ to the satisfiability problem of Σ_0 formulas which correspond to formulas in the token color logic L which is supposed to has a decidable satisfiability problem. The proof is constructive: we build from a formula in Σ_2 a formula in L which has an equivalent satisfaction problem. The complexity of this construction is given at the end of the section.

In the remaining of the section, we denote by ψ a closed formula in Σ_2 in prenex form $\exists \overrightarrow{x} \exists \overrightarrow{X} \exists \overrightarrow{m} \exists \overrightarrow{\gamma}\,.\,\varphi$, where φ is a formula in Π_1 in prenex form $\forall \overrightarrow{y} \forall \overrightarrow{Y} \forall \overrightarrow{n} \forall \overrightarrow{\alpha}\,.\,\phi$, with $\overrightarrow{x}, \overrightarrow{y} \in T$, $\overrightarrow{X}, \overrightarrow{Y} \in \mathcal{X}$, $\overrightarrow{m}, \overrightarrow{n} \in C$, $\overrightarrow{\gamma}, \overrightarrow{\alpha} \in \Gamma$, and $\phi \in \Sigma_0$. We assume that all quantified variables are different.

We prove first that the fragment Σ_2 has the small model property, i.e., every satisfiable formula ψ in Σ_2 has a model of a bounded size (where the size is the number of tokens in each place).

Lemma 1. *The Σ_2 fragment of $\mathtt{CTSL}(L)$ has a small model property.*

Proof. Suppose that there is a set coloring $\langle \nu, \mu \rangle$ which satisfies the Σ_2 formula ψ. This means that there is a set of tokens $N \subseteq \mathbb{N}$ (resp. of colors $\overrightarrow{c}$) and a mapping $\theta \in [\overrightarrow{x} \to N]$ (resp. $\delta \in [\overrightarrow{m} \to \overrightarrow{c}]$) such that $\langle \nu, \mu \rangle \models_{\theta,\delta} \varphi$.

Let N_1 be the finite subset of N corresponding to the image of the (finite set of) existential quantified token variables $\overrightarrow{x}$ through θ, i.e., $N_1 = \cup_{x \in \overrightarrow{x}} \theta(x)$. Let A be a fresh symbol in $\mathcal{X}$. Without changing the satisfiability of ψ, we strengthen it by stating that A captures exactly N_1. Indeed, $\langle \nu, \mu \rangle \models_{\theta,\delta} \varphi$ is equivalent to $\langle \nu[A \leftarrow N_1], \mu \rangle \models_{\theta,\delta} \mathit{All}(\overrightarrow{x}, A) \wedge \varphi$ where:

$$\mathit{All}(\overrightarrow{x}, A) \equiv A(\overrightarrow{x}) \wedge (\forall y.\ (\wedge_{x_i \in \overrightarrow{x}} y \neq x_i) \implies \neg A(y))$$

From φ, we obtain a weaker formula φ_A (i.e., $\varphi_A \Rightarrow \varphi$) by restricting the universal quantified variables $\overrightarrow{y}$ and $\overrightarrow{Y}$ to the set of tokens N_1 represented by A:

$$\varphi_A \equiv \forall \overrightarrow{y} \forall \overrightarrow{Y}\,.\,(\bigwedge_{y_i \in \overrightarrow{y}} A(y_i) \wedge \bigwedge_{Y_i \in \overrightarrow{Y}} \mathit{Subset}(Y_i, A)) \Rightarrow \forall \overrightarrow{n} \forall \overrightarrow{\alpha}\,.\,\phi$$

$$\text{where} \quad \mathit{Subset}(Y, A) \equiv \forall z.\ Y(z) \Rightarrow A(z)$$

Then, $\langle \nu[A \leftarrow N_1], \mu \rangle \models_{\theta,\delta} \mathit{All}(\overrightarrow{x}, A) \wedge \varphi_A$. In φ_A and $\mathit{All}(\overrightarrow{x}, A)$, all universally quantified token variables are tested to belong to the finite set A, which means that only tokens in N_1 are relevant. If $\langle \nu_1, \mu_1 \rangle$ is the restrictions of ν and μ over N_1, then $\langle \nu_1[A \leftarrow N_1], \mu_1 \rangle$ is a model for $\varphi_A \wedge \mathit{All}(\overrightarrow{x}, A)$ which is finite. But $\varphi_A \wedge \mathit{All}(\overrightarrow{x}, A) \Rightarrow \varphi$ over N_1 so $\langle \nu_1[A \leftarrow N_1], \mu_1 \rangle$ is also a model of ψ.

So, we have shown that if a Σ_2 formula ψ has a model, then ψ has also a model over a finite domain of tokens which size is bounded by the number of the existential quantified token variables in ψ (denoted here by $\overrightarrow{x}$).

Based on the above result, we are able to get rid of the universal quantifications over token and sets of token variables in Σ_2 formula by replacing them by finite conjunctions.

Lemma 2. *The satisfiability problem for the Σ_2 fragment of* CTSL *reduces to the satisfiability problem for the Σ_1 fragment.*

Proof. Let N be the finite set of tokens on which is built the finite model of the ψ formula in Σ_2, and let $\langle \nu, \mu \rangle$, θ, and δ be this finite model, i.e., $\langle \nu, \mu \rangle \models_{\theta,\delta} \forall \overrightarrow{y} \forall \overrightarrow{Y} \forall \overrightarrow{n} \forall \overrightarrow{\alpha} . \phi$ with ν, μ, and θ restricted to N. Recall that N is the image of variables in $\overrightarrow{x}$ by θ. This means that, from a syntactic point of view, the universe of variables in $\overrightarrow{y}$ and $\overrightarrow{Y}$ is fully described by variables in $\overrightarrow{x}$. Then, we can replace any universal quantification over tokens by a finite conjunction where each conjunct is obtained from φ by a substitution of variables in $\overrightarrow{y}$ with variables in $\overrightarrow{x}$:

$$\langle \nu, \mu \rangle \models_{\theta,\delta} \bigwedge_{\sigma \in [\overrightarrow{y} \to \overrightarrow{x}]} \forall \overrightarrow{Y} \forall \overrightarrow{n} \forall \overrightarrow{\alpha} . \phi[\sigma]$$

Elimination of universally quantified variables in $\overrightarrow{Y}$ is done by a slight generalization of the above construct: we guess all the possible valuations for variables in $\overrightarrow{Y}$ in terms of sets of variables in $\overrightarrow{x}$. A such valuation ξ is a mapping in $[\overrightarrow{Y} \to (\overrightarrow{x} \to \{true, false\})]$. The effect of applying ξ to a formula in CTSL is to replace all subformula $Y(x)$ by $\xi(Y)(x) \in \{true, false\}$. However, valuations ξ shall satisfy the following consistency property wrt valuation of token variables given by the mapping θ: if variables x_i and x_j in $\overrightarrow{x}$ are mapped to the same token by θ, then they have to belong to the same set in $\overrightarrow{Y}$. This consistency property is given by the following formula:

$$Consist(\overrightarrow{Y}, \overrightarrow{x}) \equiv \bigwedge_{x_i, x_j \in \overrightarrow{x}} (x_i = x_j) \Rightarrow \bigwedge_{Y \in \overrightarrow{Y}} (Y(x_i) \Leftrightarrow Y(x_j))$$

We use it as premise in each conjunct corresponding to a valuations for universally quantified variables in $\overrightarrow{Y}$:

$$\langle \nu, \mu \rangle \models_{\theta,\delta} \bigwedge_{\sigma \in [\overrightarrow{y} \to \overrightarrow{x}]} \bigwedge_{\xi \in [\overrightarrow{Y} \to (\overrightarrow{x} \to \{true, false\})]} (Consist(\overrightarrow{Y}, \overrightarrow{x}) \Rightarrow \forall \overrightarrow{n} \forall \overrightarrow{\alpha} . \phi[\sigma])[\xi]$$

The formula above has no more universal quantification over token or sets of tokens.

To eliminate the universal quantification over coloring symbols in $\overrightarrow{\alpha}$, we note that mappings in $\overrightarrow{\alpha}$ are only applied now to variables in $\overrightarrow{x}$. We define a substitution η which maps each occurrence of a term $\alpha(x)$ with $\alpha \in \overrightarrow{\alpha}$ and $x \in \overrightarrow{x}$ to a new color variable in C, given by $\eta(\alpha)(x)$. A similar consistency property as the one used for the elimination second order of variables in $\overrightarrow{Y}$ has to be satisfied by the substitution η: if two variables x_i and x_j are mapped to the same token, then the substitution η shall map them to the same color for any coloring symbol in $\overrightarrow{\alpha}$. We can combine these properties and obtain the formula:

$$Consist'(\overrightarrow{Y}, \overrightarrow{\alpha}, \overrightarrow{x}) \equiv \bigwedge_{x_i, x_j \in \overrightarrow{x}} (x_i = x_j) \Rightarrow \bigwedge_{Y \in \overrightarrow{Y}} (Y(x_i) \Leftrightarrow Y(x_j)) \wedge \bigwedge_{\alpha \in \overrightarrow{\alpha}} (\alpha(x_i) = \alpha(x_j))$$

Then, if $\overrightarrow{a} \subset C$ is the image of η, the satisfaction of ψ is equivalent to the following satisfaction problem:

$$\langle \nu, \mu \rangle \models_{\theta,\delta} \bigwedge_{\sigma \in [\overrightarrow{y} \to \overrightarrow{x}]} \bigwedge_{\xi \in [\overrightarrow{Y} \to (\overrightarrow{x} \to \{true, false\})]} \forall \overrightarrow{n}, \overrightarrow{a} . (Consist'(\overrightarrow{Y}, \overrightarrow{\alpha}, \overrightarrow{x}) \Rightarrow \phi[\sigma])[\xi][\eta]$$

which concerns a Σ_1 formula.

Finally, we eliminate the existential quantifiers over tokens, sets of tokens, and coloring symbols variables by introducing new existential quantifiers on color variables.

Lemma 3. *The satisfiability problem for the Σ_1 fragment of $\mathtt{CTSL}(L)$ reduces to the satisfiability problem for L.*

Proof. Let ψ be a Σ_1 formula in $\mathtt{CTSL}(L)$ of the prenex form $\exists \overrightarrow{x} \exists \overrightarrow{X} \exists \overrightarrow{\gamma} . \varphi$ with $\varphi \in L$. From Lemma 1 we know that ψ is satisfiable iff it exists a model $\langle \nu, \mu \rangle$ and mappings θ and δ defined on a set of tokens N which size is at most $n = |\overrightarrow{x}|$.

We build from ψ a formula in L which has an equivalent satisfiability problem. First, we transform ψ in order to eliminate atomic subformulas corresponding to equality of token variables, and so to obtain a Σ_1 formula where all quantified token variables are distinct. For this, let $\mathcal{B}(\overrightarrow{x})$ be the set of mappings $b \in [\overrightarrow{x} \to \overrightarrow{x}]$ representing an equivalence relation between variables in $\overrightarrow{x}$. (The size of $\mathcal{B}(\overrightarrow{x})$ is less than n^n but has no simple formulation.) For any $b \in \mathcal{B}(\overrightarrow{x})$, $b(x_i) = x_j$ if x_j uniquely represents the equivalence class of x_i. We denote by $\overrightarrow{x}_b = Img(b)$ the set of variables representing equivalence classes in b. A mapping b applied to a formula $\varphi(\overrightarrow{x})$, denoted by $\varphi[b]$, substitutes each occurence of a variable $x \in \overrightarrow{x}$ by $b(x)$ and replaces all atomic formulas $x_i = x_j$ by true if $b(x_i) = b(x_j)$ and by false otherwise. Then, ψ is satisfiable iff the following formula is satisfiable:

$$\bigvee_{b \in \mathcal{B}(\overrightarrow{x})} \exists \overrightarrow{x}_b \exists \overrightarrow{X} \exists \overrightarrow{\gamma} . \varphi[b]$$

The problem is now to reduce the satisfiability of a Σ_1 formula where all quantified token variables range over distinct tokens to the satisfiability of an L formula.

To eliminate existential quantifiers over sets of tokens $\overrightarrow{X}$, we use the finite model property in a similar way than in Lemma 2: we consider all valuations of variables in $\overrightarrow{X}$ in terms of sets of variables in $\overrightarrow{x}_b$. Since all variables in $\overrightarrow{x}_b$ are distinct, we don't need the consistency property. Then, ψ is satisfiable iff the following formula is satisfiable:

$$\bigvee_{b \in \mathcal{B}(\overrightarrow{x})} \bigvee_{\xi \in [\overrightarrow{X} \to (\overrightarrow{x}_b \to \{true, false\})]} \exists \overrightarrow{x}_b \exists \overrightarrow{\gamma} . (\varphi[b])[\xi]$$

The second order quantifiers on color symbols $\overrightarrow{\gamma}$ are eliminating like in the proof of Lemma 2. We define a substitution η which maps each occurrence of a term $\gamma(x)$ with $\gamma \in \overrightarrow{\gamma}$ and $x \in \overrightarrow{x}_b$ to a new color variable in C, given by $\eta(\gamma)(x)$. If $\overrightarrow{a} \subset C$ is the image of η, then ψ is satisfiable iff the following formula is satisfiable:

$$\bigvee_{b \in \mathcal{B}(\overrightarrow{x})} \bigvee_{\xi \in [\overrightarrow{X} \to (\overrightarrow{x}_b \to \{true, false\})]} \exists \overrightarrow{x}_b \exists \overrightarrow{a} . (\varphi[b])[\xi][\eta]$$

Note that mappings b, ξ, and η replace all occurences of token variables in $\overrightarrow{x}_b$ by constant formula (*true*, *false*) or color variables. Then $\varphi[b][\xi][\eta]$ does not contain any occurence of variables in $\overrightarrow{x}_b$, so the $\exists \overrightarrow{x}_b$ can be eliminated, QED.

Complexity of satisfiability checking: The proof above builds from a formula ψ in Σ_2 an equivalent formula ψ_0 in Σ_0. Let ψ be the prenex form formula $\exists\overrightarrow{x}\exists\overrightarrow{X}\exists\overrightarrow{m}\exists\overrightarrow{\gamma}.\ \varphi$, where φ is a formula in Π_1 in prenex form $\forall\overrightarrow{y}\forall\overrightarrow{Y}\forall\overrightarrow{n}\forall\overrightarrow{\alpha}.\ \phi$. The size of the formula ψ_0 depends on the size of ψ as follows:

The reduction proposed in the proof of Lemma 2 eliminates $\forall\overrightarrow{y}$ (resp. $\forall\overrightarrow{Y}$) quantifiers by introducing $|\overrightarrow{x}|^{|\overrightarrow{y}|}$ (resp. $|\overrightarrow{Y}|^{2^{|\overrightarrow{x}|}}$) conjuncts. Each conjunct contains formula ϕ and a formula *Consist'* which contains $|\overrightarrow{x}|^2.(|\overrightarrow{Y}|+|\overrightarrow{\alpha}|)$ atomic formula. The number of quantified color variables added by η is equal to is equal to $|\overrightarrow{\alpha}|.|\overrightarrow{x}|$. We obtain from φ a Σ_0 formula φ_0 which is prefixed by at least $|\overrightarrow{\alpha}|.|\overrightarrow{x}|+|\overrightarrow{n}|$ universal quantifiers over color variables and which size is $|\overrightarrow{x}|^{|\overrightarrow{y}|}.|\overrightarrow{Y}|^{2^{|\overrightarrow{x}|}}.(|\phi|+|\overrightarrow{x}|^2.(|\overrightarrow{Y}|+|\overrightarrow{\alpha}|))$. Let call ψ_1 the formula $\exists\overrightarrow{x}\exists\overrightarrow{X}\exists\overrightarrow{m}\exists\overrightarrow{\gamma}.\ \varphi_0$.

ψ_1 is further reduced by the proof of Lemma 3 into a Σ_0 formula by eliminating $\exists\overrightarrow{x}$ and $\exists\overrightarrow{X}$ quantifiers. The reduction adds at most $|\overrightarrow{x}|^{|\overrightarrow{x}|}.|\overrightarrow{X}|^{2^{|\overrightarrow{x}|}}$ disjuncts (the first term is an over approximation of the size of $\mathcal{B}(\overrightarrow{X})$). Each disjunct is built from ψ_1 prefixed by at most $|\overrightarrow{m}|.|\overrightarrow{\gamma}|.|\overrightarrow{x}|$ existential quantifiers. We obtain a Σ_0 formula ψ_0 which number of atomic subformulais $O(|\overrightarrow{x}|^{|\overrightarrow{x}|+|\overrightarrow{y}|}.|\overrightarrow{Y}|^{2^{|\overrightarrow{x}|}}.|\overrightarrow{X}|^{2^{|\overrightarrow{x}|}})$,greater than the one of ψ and which is prefixed by at most $|\overrightarrow{x}|.(|\overrightarrow{\alpha}|+|\overrightarrow{\gamma}|)$ new quantifiers.

Note that each reduction step has been considered above in its worst combinatorial case assumptions. However, the initial formula comes usually with syntactic properties that reduce this high complexity.

For example, constraints in program models and invariant properties are usually first-order formulas (see examples in the appendix). Then, post-images do not involve second order universal quantification and the decision procedure is applied in this case to formulas of the form $\exists\overrightarrow{x}\exists\overrightarrow{X}\exists\overrightarrow{m}\exists\overrightarrow{\gamma}.\ \forall\overrightarrow{y}\forall\overrightarrow{n}.\ \varphi$. Moreover, the existentially quantified second order variables appearing in the post-images are all disjoint because they are included in disjoint places of $\mathcal{P}$. This implies a one-to-one mapping between variables in $\overrightarrow{x}$ and those in $\overrightarrow{X}$. This mapping is further improved if variables in $\overrightarrow{x}$ have explicitely specified places (which is true if they correspond to variables of the existential part of a transition), which reduces the combinatoric explosion due to case splitting according to all possible locations of tokens in places. Then, the decision procedure of the color logic is invoked on a formula which is prefixed by $|\overrightarrow{\gamma}|.|\overrightarrow{x}|$ new existential quantification and is $O(|\overrightarrow{x}|^{|\overrightarrow{x}|+|\overrightarrow{y}|})$ times greater in size than φ. Finally, transitions of CTN have only few (2 or 3) variables in the set $\overrightarrow{x}$ (see examples) since local synchronization involve in general two processes. Hence, the growing factor $|\overrightarrow{x}|^{|\overrightarrow{x}|+|\overrightarrow{y}|}$ is small in practice.

B A logic for reasoning about stacks

Processes with procedure calls can be modeled using pushdown systems. Therefore, tokens representing occurrences of such processes must be colored by the contents of their stacks. Let us consider the case where the alphabet of these stack is finite. Then, stacks can be represented as finite words over this alphabet, and constraints over stack contents can be defined using logics over finite words. We introduce hereafter such a logic called SRC (for Synchronous Rational Constraints).

Let Σ be a finite alphabet, and let $\mathsf{Reg}(\Sigma)$ be the class of regular languages over Σ. Then, we consider the following relations:

- let $\leq_{\mathsf{pref}}$ be the prefix ordering between words, i.e., for every $u, v \in \Sigma^*$. $u \leq_{\mathsf{pref}} v$ iff $\exists w.\ v = uw$,
- let $=_\ell$ be the length equality predicate i.e., for every $u, v \in \Sigma^*$, $u =_\ell v$ iff $|u| = |v|$, and
- for every $R \in \mathsf{Reg}(\Sigma)$, let $R(\cdot)$ be the unary predicate such that for every $u \in \Sigma^*$, $R(u)$ iff $u \in R$.

Notice that the equality predicate can be defined using the relations $\leq_{\mathsf{pref}}$ and $=_\ell$: $u = v$ is equivalent to $u \leq_{\mathsf{pref}} v \wedge u =_\ell v$.

Then, let $\mathsf{SRC} = (\Sigma^*, \{\lambda u.\, u \cdot a \ :\ a \in \Sigma\}, \{\leq_{\mathsf{pref}}, =_\ell\} \cup \{R(\cdot)\ :\ R \in \mathsf{Reg}(\Sigma)\})$.

Theorem 6. *The satisfiability problem of* SRC *is decidable.*

Proof. (Sketch) Given a formula $\phi(z_1, \ldots, z_n)$, let $[\![\phi]\!]$ be the set of vectors of words $w \in (\Sigma^*)^n$ such that $\delta \models_{\mathsf{SRC}} \phi$ where δ is the valuation such that, for every $i \in \{1, \ldots, n\}$, $\delta(z_i) = w_i$. We prove that for every formula ϕ with n free variables, the set $[\![\phi]\!]$ is a n-dim rational set, i.e., definable by a n-tape finite-state automaton (transducer), for which the emptyness problem is of course decidable. For that, we show that all atomic formulas define *synchronous* rational sets, and we use the fact that the class of synchronous rational sets is closed under boolean operations and projection (see [16]).

C Example: Accessing Multiple Shared Resources

Let us consider a concurrent Java application where an infinite set of threads are sharing the objects in the set $O = \{o, o_1, o_2, o_3, o_4\}$. Suppose that threads are executing non-deterministically two tasks, each task acquiring a subset of the shared objects by calling synchronized methods of these objects.

For example, consider the ICFG given in the listing L below:

```
   task1() {
1:    o1.m1();
2: }
   task2() {
3:    o2.m1();
4: }
   synchronized m1() {
5:    o.m();
6: }
   synchronized m2() {
7:    o.n();
8: }
   synchronized m() {
9:    o3.m3();
10:}
```

```
  synchronized n() {
11:   o4.m4();
12:}
  synchronized m3() {
13:   o4.n4();
14:}
  synchronized n3() {
15:   /* some action on o2, o, o3, o4 */
      /* but not wait or notify */
    }
  synchronized m4() {
16:   o3.n3();
17:}
  synchronized n4() {
18:   /* some action on o1, o, o3, o4 */
      /* but not wait or notify */
    }
```

This example does not respect the simple rule of acquiring shared objects (here $\{o, o_3, o_3\}$) in the same order. So it is potentially generator of deadlocks. We will show in the following that our methodology is able to prove absence of deadlocks for this example.

Since this ICFG does not call any of `wait`, `notify`, or `notifyAll` methods, places $susp_i$ (with $i \in O$) are not connected in the corresponding model. So, we do not add these places in our model, nor consider the rules corresponding to the translation of the call of these methods.

The CTN[SRC] obtained is given in Table 1.

To this model, we have to add the following assertions saying that places corresponding to locks have at most one token:

$$Lock \equiv \wedge_{p \in \{lock_{o_1}, \ldots, lock_{o_4}} \forall t, t'.\ p(t) \wedge p(t') \implies t = t'$$

On this model, we want to verify the following properties:

- If it exists a thread at control point l_9, there is no thread having the lock on object o_3:
$$\phi(l_9, o_3) \equiv \forall t, t'.\ l_9(t) \implies \neg(\alpha(t') \in \Sigma^*.o_3.\Sigma^*)$$
This property ensures that when a thread tries to acquire o_3, it will acquire it immediately and cannot be delayed by a thread owning the lock of this object.
The same property is true for pairs $(l_{11}, o_4), (l_{13}, o_4), (l_{15}, o_3)$. Then, the conjuction of these properties says that it is not possible to have deadlock when acquiring object o_3 and o_4, though they are not acquired in the same order by all tasks.
- There is at most one thread waiting to obtain the lock on object o:
$$\forall t, t'.\ wait_o(t) \wedge wait_o(t') \implies t = t'$$
This property ensures that a single thread executing `taski` ($i = 1..2$) wis trying to lock object o.

$$
\begin{array}{lrcll}
m1^1_{o_1}: & (x,y).\ (l_1, lock_{o_1}) & \hookrightarrow & (l_5,\bot) & : \alpha'(x) = \alpha(x).l_2.o_1 \\
m1^2_{o_1}: & x.\ l_1 & \hookrightarrow & l_5 & : \alpha(x) \in (\Sigma^*.o_1.\Sigma^*) \wedge \alpha'(x) = \alpha(x).l_2.o_1 \\
m1^3_{o_1}: & x.\ l_1 & \hookrightarrow & wait_{o_1} & \\
 & & & & : \neg(\exists y.\ lock_{o_1}(y)) \wedge \neg(\alpha(x) \in (\Sigma^*.o_1.\Sigma^*)) \wedge \alpha'(x) = \alpha(x).l_2.l_5 \\
m1^4_{o_1}: & (x,y).\ (wait_{o_1}, lock_{o_1}) & \hookrightarrow & (l_5,\bot) & : \exists w.\ \alpha(x) = w.l_5 \wedge \alpha'(x) = w.o_1 \\
 & & & & \\
m2^1_{o_2}: & (x,y).\ (l_3, lock_{o_2}) & \hookrightarrow & (l_7,\bot) & : \alpha'(x) = \alpha(x).l_4.o_2 \\
m2^2_{o_2}: & x.\ l_3 & \hookrightarrow & l_7 & : \alpha(x) \in (\Sigma^*.o_2.\Sigma^*) \wedge \alpha'(x) = \alpha(x).l_4.o_2 \\
m2^3_{o_2}: & x.\ l_3 & \hookrightarrow & wait_{o_2} & \\
 & & & & : \neg(\exists y.\ lock_{o_2}(y)) \wedge \neg(\alpha(x) \in (\Sigma^*.o_2.\Sigma^*)) \wedge \alpha'(x) = \alpha(x).l_4.l_7 \\
m2^4_{o_2}: & (x,y).\ (wait_{o_2}, lock_{o_2}) & \hookrightarrow & (l_7,\bot) & : \exists w.\ \alpha(x) = w.l_7 \wedge \alpha'(x) = w.o_2 \\
 & & & & \\
m^1_o: & (x,y).\ (l_5, lock_o) & \hookrightarrow & (l_9,\bot) & : \alpha'(x) = \alpha(x).l_6.o \\
m^2_o: & x.\ l_5 & \hookrightarrow & l_9 & : \alpha(x) \in (\Sigma^*.o.\Sigma^*) \wedge \alpha'(x) = \alpha(x).l_6.o \\
m^3_o: & x.\ l_5 & \hookrightarrow & wait_o & \\
 & & & & : \neg(\exists y.\ lock_o(y)) \wedge \neg(\alpha(x) \in (\Sigma^*.o.\Sigma^*)) \wedge \alpha'(x) = \alpha(x).l_6.l_9 \\
m^4_o: & (x,y).\ (wait_o, lock_o) & \hookrightarrow & (l_9,\bot) & : \exists w.\ \alpha(x) = w.l_9 \wedge \alpha'(x) = w.o \\
 & & & & \\
n^1_o: & (x,y).\ (l_7, lock_o) & \hookrightarrow & (l_{11},\bot) & : \alpha'(x) = \alpha(x).l_8.o \\
n^2_o: & x.\ l_7 & \hookrightarrow & l_{11} & : \alpha(x) \in (\Sigma^*.o.\Sigma^*) \wedge \alpha'(x) = \alpha(x).l_8.o \\
n^3_o: & x.\ l_7 & \hookrightarrow & wait_o & \\
 & & & & : \neg(\exists y.\ lock_o(y)) \wedge \neg(\alpha(x) \in (\Sigma^*.o.\Sigma^*)) \wedge \alpha'(x) = \alpha(x).l_8.l_{11} \\
n^4_o: & (x,y).\ (wait_o, lock_o) & \hookrightarrow & (l_{11},\bot) & : \exists w.\ \alpha(x) = w.l_{11} \wedge \alpha'(x) = w.o \\
 & & & & \\
m3^1_{o_3}: & (x,y).\ (l_9, lock_{o_3}) & \hookrightarrow & (l_{13},\bot) & : \alpha'(x) = \alpha(x).l_{10}.o_3 \\
m3^2_{o_3}: & x.\ l_9 & \hookrightarrow & l_{13} & : \alpha(x) \in (\Sigma^*.o_3.\Sigma^*) \wedge \alpha'(x) = \alpha(x).l_{10}.o_3 \\
m3^3_{o_3}: & x.\ l_9 & \hookrightarrow & wait_{o_3} & \\
 & & & & : \neg(\exists y.\ lock_{o_3}(y)) \wedge \neg(\alpha(x) \in (\Sigma^*.o_3.\Sigma^*)) \wedge \alpha'(x) = \alpha(x).l_{10}.l_{13} \\
m3^4_{o_3}: & (x,y).\ (wait_{o_3}, lock_{o_3}) & \hookrightarrow & (l_{13},\bot) & : \exists w.\ \alpha(x) = w.l_{13} \wedge \alpha'(x) = w.o_3 \\
 & & & & \\
m4^1_{o_4}: & (x,y).\ (l_{11}, lock_{o_4}) & \hookrightarrow & (l_{16},\bot) & : \alpha'(x) = \alpha(x).l_{12}.o_4 \\
m4^2_{o_4}: & x.\ l_{11} & \hookrightarrow & l_{16} & : \alpha(x) \in (\Sigma^*.o_4.\Sigma^*) \wedge \alpha'(x) = \alpha(x).l_{12}.o_4 \\
m4^3_{o_4}: & x.\ l_{11} & \hookrightarrow & wait_{o_4} & \\
 & & & & : \neg(\exists y.\ lock_{o_4}(y)) \wedge \neg(\alpha(x) \in (\Sigma^*.o_4.\Sigma^*)) \wedge \alpha'(x) = \alpha(x).l_{12}.l_{16} \\
m4^4_{o_4}: & (x,y).\ (wait_{o_4}, lock_{o_4}) & \hookrightarrow & (l_{16},\bot) & : \exists w.\ \alpha(x) = w.l_{16} \wedge \alpha'(x) = w.o_4 \\
 & & & & \\
n4^1_{o_4}: & (x,y).\ (l_{13}, lock_{o_4}) & \hookrightarrow & (l_{18},\bot) & : \alpha'(x) = \alpha(x).l_{14}.o_4 \\
n4^2_{o_4}: & x.\ l_{13} & \hookrightarrow & l_{18} & : \alpha(x) \in (\Sigma^*.o_4.\Sigma^*) \wedge \alpha'(x) = \alpha(x).l_{14}.o_4 \\
n4^3_{o_4}: & x.\ l_{13} & \hookrightarrow & wait_{o_4} & \\
 & & & & : \neg(\exists y.\ lock_{o_4}(y)) \wedge \neg(\alpha(x) \in (\Sigma^*.o_4.\Sigma^*)) \wedge \alpha'(x) = \alpha(x).l_{14}.l_{18} \\
n4^4_{o_4}: & (x,y).\ (wait_{o_4}, lock_{o_4}) & \hookrightarrow & (l_{18},\bot) & : \exists w.\ \alpha(x) = w.l_{18} \wedge \alpha'(x) = w.o_4 \\
 & & & & \\
n3^1_{o_3}: & (x,y).\ (l_{16}, lock_{o_3}) & \hookrightarrow & (l_{15},\bot) & : \alpha'(x) = \alpha(x).l_{17}.o_3 \\
n3^2_{o_3}: & x.\ l_{16} & \hookrightarrow & l_{15} & : \alpha(x) \in (\Sigma^*.o_3.\Sigma^*) \wedge \alpha'(x) = \alpha(x).l_{17}.o_3 \\
n3^3_{o_3}: & x.\ l_{16} & \hookrightarrow & wait_{o_3} & \\
 & & & & : \neg(\exists y.\ lock_{o_3}(y)) \wedge \neg(\alpha(x) \in (\Sigma^*.o_3.\Sigma^*)) \wedge \alpha'(x) = \alpha(x).l_{17}.l_{15} \\
n3^4_{o_3}: & (x,y).\ (wait_{o_3}, lock_{o_3}) & \hookrightarrow & (l_{15},\bot) & : \exists w.\ \alpha(x) = w.l_{15} \wedge \alpha'(x) = w.o_3
\end{array}
$$

Table 1. Model of the ICFG of listing L.

D Example: Fischer's Protocol

The protocol [24] is to guarantee mutual exclusion in a concurrent system consisting of any number of processes. It assumes only atomic reads and writes and the mutual exclusion relies heavily on timing constraints associated with the execution of instructions.

D.1 Protocol program

Informally, the protocol proceeds as follows. A global variable g is used to store identities of process or 0. Its value is initialy 0. When a process i wants to access to the critical section, it waits to see g at 0. Then, it assigns g to its identity. After that, it waits for some time (greater than the time taken by assignment instructions), and if g has a value equal to its identity after this time, it is safe to enter the critical section. The pseudo-C code executed by each process is the following:

```
do {
  do {
    await (x==0);
    x = 1; // takes time in [A,A']
    delay; // takes time in [D,D']
  } while (x!=id); // atomic test
  critical_section;
  x = 0;   // takes time in [A,A']
} while (1);
```

D.2 CTN model

The model of this protocol using $\mathtt{CTN}[\mathsf{FO}_{\mathbb{R}}]$ is given in Table 2. We model each process by a token. Each token is assumed to have two local clocks (a and d) measuring the time taken by assignment resp. delay instructions, and an identity (id). Our model is parameterized in two dimensions. First, the number of processes is not fixed. Second, we suppose that an assignment takes between $[A,A']$ time units and the delay instruction between $[D,D']$ time units. Test instructions take no time.

D.3 Mutual exclusion property and its proof

It has been proven that for a finite number of processes, Fischer's protocol ensures the mutual exclusion property if $0 \leq A \leq A' < D \leq D'$. The mutual exclusion property is expressed by the following Π_1 formula:

$$CS \equiv \forall x,x'.\ cs(x) \wedge cs(x') \implies x = x'$$

which, due to the unicity of identities, it is a consequence of the following simpler formula:

$$CS' \equiv \forall t.\ cs(t) \implies g = id(t)$$

$$
\begin{array}{lll}
try: & x.\ idle \hookrightarrow try & \\
 & \quad : g = 0 \wedge a'(x) = 0 \wedge id(x) = id'(x) \wedge g' = g \wedge d(x) = d'(x) & \\
set: & x.\ try \hookrightarrow wait & \\
 & \quad : a(x) \geq A \wedge g' = id(x) \wedge d'(x) = 0 \wedge id(x) = id'(x) \wedge a(x) = a'(x) & \\
enter: & x.\ wait \hookrightarrow cs & \\
 & \quad : d(x) \geq D \wedge g = id(x) \wedge id(x) = id'(x) \wedge g' = g \wedge d(x) = d'(x) \wedge a(x) = a'(x) & \\
retry: & x.\ wait \hookrightarrow idle & \\
 & \quad : d(x) \geq D \wedge g \neq id(x) \wedge id(x) = id'(x) \wedge g' = g \wedge d(x) = d'(x) \wedge a(x) = a'(x) & \\
exit: & x.\ cs \hookrightarrow idle & \\
 & \quad : g' = 0 \wedge id(x) = id'(x) \wedge d(x) = d'(x) \wedge a(x) = a'(x) & \\
time: & \quad \hookrightarrow : \exists z.\ 0 < z \wedge t' = t + z \wedge g' = g & \\
 & \quad | \ \ (i,r,w,c).\ (idle, try, wait, cs) \mapsto (idle, try, wait, cs) & \\
 & \qquad : a'(i) = a(i) + (t' - t) \wedge d'(i) = d(i) + (t' - t) \wedge id'(i) = id(i) & \\
 & \qquad \wedge a'(r) = a(r) + (t' - t) \wedge d'(r) = d(r) + (t' - t) \wedge id'(r) = id(r) & \\
 & \qquad \wedge a'(w) = a(w) + (t' - t) \wedge d'(w) = d(w) + (t' - t) \wedge id'(w) = id(w) & \\
 & \qquad \wedge a'(c) = a(c) + (t' - t) \wedge d'(c) = d(c) + (t' - t) \wedge id'(c) = id(c) & \\
 & \qquad \wedge (\forall z'.\ 0 \leq z' \leq (t' - t) \implies a(r) + z' \leq A' \wedge d(w) + z' \leq D') &
\end{array}
$$

Table 2. `CTN` model for the Fischer's protocol.

To prove invariance of CS' for our model, we use our verification methodology to show that the following formula I (in fragment Π_1) is an inductive invariant:

$$
\begin{array}{rl}
 & I \equiv CS' \wedge I_p \wedge Inv(try) \wedge Inv(wait) \wedge I_{try} \wedge I_{wait} \wedge I_{cs} \\
\text{where} & \\
I_p & \equiv 0 \leq A \leq A' < D \leq D' \\
Inv(try) & \equiv \forall t.\ try(t) \implies a(t) \leq A' \\
Inv(wait) & \equiv \forall t.\ wait(t) \implies d(t) \leq D' \\
I_{try} & \equiv (\forall t, t'.\ try(t) \wedge try(t') \implies |a(t) - a(t')| \leq A') \\
 & \quad \wedge\ (\forall t, t'.\ try(t) \wedge wait(t') \implies |a(t) - a(t')| \leq A') \\
I_{wait} & \equiv (\forall t, t'.\ wait(t) \wedge wait(t') \implies (|a(t) - a(t')| \leq A' \wedge |d(t) - d(t')| \leq A')) \\
 & \quad \wedge\ (\forall t.\ wait(t) \implies A \leq a(t) - d(t) \leq A') \\
 & \quad \wedge\ (\forall t, t'.\ wait(t) \wedge d(t) \geq D \implies \neg try(t)) \\
I_{cs} & \equiv (\forall t, t'.\ cs(t) \Leftrightarrow \neg try(t')) \\
 & \quad \wedge\ (\forall t.\ g = 0 \implies \neg(cs(t) \vee wait(t)))
\end{array}
$$

Intuitively, invariant I_{try} expresses the property that the time between the arrival of two processes in *try* state may not be greater than A'. This property is also true for any pair of processes, one in the *try* state and the other in the *wait* state. In the proof, I_{try} is useful to establish I_{wait} which says that the difference between the arrival times of processes in the *wait* state is also bounded by A'. This implies that (see last conjunct of I_{wait}), when processes stay more than $D > A'$ in state *wait*, there is no more process in state *try*. Then, when a process enters in state *cs*, no process are in *try* state to change

global variable g and so to allow another process in *wait* state to enter in *cs*. This last property is expressed by invariant I_{cs}.

For all discrete transitions (i.e., with an empty universal part), we can compute the post image of the transition on our invariant using the simplified computation defined in [11] based on the the first order version of CML.

Transition try: The post-image of I by transition *try* is given by:

$$\begin{aligned}\mathsf{post}_{try}(I) \equiv \exists y.\ \exists id_x, d_x a_x.\ \exists g^-.\ & (I[g \leftarrow g^-] \ominus (x \mapsto idle) \oplus (y \mapsto try)) \wedge \\ & g^- = 0 \wedge a(y) = 0 \wedge id_x = id(y) \wedge g = g^- \wedge d_x = d(y)\end{aligned}$$

The parts of I changed by post are $Inv(try)$ and I_{try}. $Inv(try)$ is true for y because $a(y) = 0$. For I_{try}, the proof uses first the fact that $g = 0$ to obtain that $\forall t.\ \neg wait(t)$ from I_{cs}. So, the second conjuct of I_{try} is trivial for y. The first conjuct is a consequence of the invariant I_{try} and of $a(y) = 0$.

Transition set: The post-image of I by transition *set* is given by:

$$\begin{aligned}\mathsf{post}_{set}(I) \equiv \exists y.\ \exists id_x, d_x a_x.\ \exists g^-.\ & (I[g \leftarrow g^-] \ominus (x \mapsto try) \oplus (y \mapsto wait)) \wedge \\ & a_x \geq A \wedge g = id_x \wedge d(y) = 0 \wedge id_x = id(y) \wedge a_x = a(y)\end{aligned}$$

Invariants changed by the post image above are: $Inv(try)$, $Inv(wait)$, I_{try}, and I_{wait}. $Inv(try)$ and $Inv(wait)$ are trivially conserved by this transition since $d(y) = 0$.

I_{try} is kept true because $a_x = a(y)$ and (from first conjunct of I_{try} instantiated for s) $\forall t'.\ |a_x - a(t')| \leq A'$. So the second conjuct of I_{try} is also satisfied for y.

The second conjunct of I_{wait} is satisfied due to the fact that $A' \geq a_x = a(y) \geq A$ and $d(y) = 0$. The third conjunct is trivially true for y (the premise is false). The first conjunct is conserved for y because:

1. By I_{try} for x says that $\forall t'.\ wait(t') \implies |a_x - a(t')| \leq A'$
2. By $a_x = a(y)$, it results that $\forall t'.\ wait(t') \implies |a(y) - a(t')| \leq A'$, so $\forall t'.\ wait(t') \implies a(t') \leq a(y) + A'$,
3. By I_{wait}, we have that $\forall t'.\ wait(t') \implies A \leq a(t') - d(t') \leq A'$ so $\forall t'.\ wait(t') \implies a(t') - A' \leq d(t') \leq a(t') - A$
4. Combining the last two results, we have that $\forall t'.\ wait(t') \implies d(t') \leq a(y) - A + A' \leq A'$ which is the first conjunct of I_{wait} for $t = y$.

Transition retry: The post-image of I by transition *retry* is given by:

$$\begin{aligned}\mathsf{post}_{retry}(I) \equiv \exists y.\ \exists id_x, d_x a_x.\ \exists g^-.\ & (I[g \leftarrow g^-] \ominus (x \mapsto wait) \oplus (y \mapsto idle)) \wedge \\ & d_x \geq D \wedge g^- \neq id_x \wedge id_x = id(y) \wedge g^+ = g^- \wedge d_x = d(y) \wedge a_x = a(y)\end{aligned}$$

Since none of the part of I concern the *idle* place where the token is added, all invariants are conserved trivially by this transition.

Transition enter: The post-image of I by transition *enter* is given by:

$$\begin{aligned}\mathsf{post}_{enter}(I) \equiv \exists y.\, \exists id_x, d_x a_x.\, \exists g^-.\ & (I[g \leftarrow g^-] \ominus (x \mapsto wait) \oplus (y \mapsto cs)) \wedge \\ & d_x \geq D \wedge g^- = id_x \wedge id_x = id(y) \wedge g^+ = g^- \wedge d_x = d(y) \wedge a_x = a(y)\end{aligned}$$

The only affected part of I are I_{cs} and CS' for which we should show that it is satisfied by y.

CS' is true by the guard of the transition $g^- = id_x = id(y) = g^+ = g$.

I_{cs} is true due to the third conjunct of I_{wait} applied to x which gives $\forall t'.\ \neg try(t')$. The second conjunct is trivially true because $g = id(y) \neq 0$.

Transition time: The post-image of I by transition *time* can be computed using the $reach_{tau}$ formula given in section 3. However, the folowing property allows us to simplify this formula due to the fact that the *time* transition satisfies the conditions of the property.

Property 1. Let τ be a CTN transition satisfying the following properties:

- the existential part does not move tokens, i.e., $n = |\overrightarrow{x}| = 0$, but φ may be not empty,
- the universal part transfers all the tokens of all places in $\mathcal{P}$ to the same places, i.e., $\overrightarrow{r} = \overrightarrow{s} = \mathcal{P} - \{\perp\}$ and ψ does not select tokens in $\overrightarrow{r}$, and
- the color of all tokens transferred is changed independently of the token place.

Then, the $reach_\tau$ formula simplifies to:

$$reach_\tau(\mathcal{P}, \mathcal{P}', \mathcal{G}, \mathcal{G}', \mathcal{L}, \mathcal{L}') = \varphi \wedge (\forall u.\, \psi)$$

Then, the post-image of the invariant becomes:

$$\begin{aligned}\mathsf{post}_{time}(I) \equiv\ & (\exists g \exists t \exists a, d, id \exists z \forall u. \\ & I \wedge 0 < z \wedge t' = t + z \wedge g' = g \\ & \wedge a'(u) = a(u) + (t' - t) \wedge d'(u) = d(u) + (t' - t) \wedge id'(u) = id(u) \\ & \wedge (\forall z'.\, 0 \leq z' \leq (t' - t) \implies (try(u) \implies a(u) + z' \leq A')) \wedge \\ & \qquad\qquad\qquad\qquad (wait(u) \implies d(u) + z' \leq D') \\ &)[g' \leftarrow g, t' \leftarrow t, a' \leftarrow a, d' \leftarrow d, id' \leftarrow id]\end{aligned}$$

Due to the fact that coloring symbols corresponding to clocks are evolving with the same quantity, the invariant I also is conserved by transition *time*.